

Taller 2 - Sistemas operativos



**Juliana Aguirre Ballesteros
Juan Carlos Santamaria Orjuela**

**Pontificia Universidad Javeriana
Facultad de ingeniería
Departamento de ingeniería de sistemas
Bogotá D.C.
2025**

Informe Taller

Introducción

El presente informe documenta el desarrollo y análisis del Taller de FORK () correspondiente a la asignatura Sistemas Operativos. Este taller implementa un conjunto de programas que demuestran la capacidad del sistema para manejar procesos concurrentes, realizar operaciones de lectura de archivos y efectuar cálculos sobre los datos leídos de dichos archivos.

El desarrollo del taller constituye una aplicación práctica de conceptos fundamentales de los sistemas operativos, tales como la comunicación entre procesos, el uso de pipes, la gestión de memoria dinámica y la creación de procesos concurrentes mediante la función fork ().

Objetivo

El objetivo del taller consiste en implementar un sistema que lea dos archivos de texto que contienen arreglos de números enteros, calcule las sumas individuales y la suma total, y que utilice procesos concurrentes para realizar dichas operaciones, comunicando los resultados entre ellos mediante un pipe.

El sistema desarrollado cumple las siguientes funciones principales:

1. **Lectura de archivos:** El sistema realiza la lectura de dos archivos de texto que contienen arreglos de enteros. Cada archivo es procesado para almacenar su contenido en memoria dinámica, validando los datos leídos antes de proceder con los cálculos.
2. **Operación de suma:** Una vez cargados los datos, el sistema calcula la suma de los elementos de cada arreglo, así como la suma total resultante de ambos. Estas operaciones se ejecutan a través de funciones para garantizar eficiencia y reutilización del código.
3. **Gestión de procesos concurrentes:** El sistema implementa la creación de procesos concurrentes mediante la función fork (). Un proceso hijo realiza la suma total, otro hijo calcula la suma del segundo arreglo, y un proceso nieto, generado por el primer hijo, calcula la suma del primer arreglo.
Esta estructura de ejecución del código permite demostrar la capacidad del sistema operativo para ejecutar múltiples procesos de manera simultánea.
4. **Comunicación entre procesos:** Los resultados generados por los procesos hijos y el nieto se transmiten al proceso principal mediante un pipe que es el mecanismo que permite la comunicación segura y unidireccional entre procesos sin compartir memoria. El proceso padre recopila los resultados y los presenta al usuario.
5. **Manejo de errores:** El sistema incluye validaciones en cada fase del proceso que son la apertura de archivos, asignación de memoria dinámica, creación de procesos y lectura de datos, en caso de error, el sistema emite un mensaje y finaliza , evitando fugas de memoria o interrupciones inesperadas.

Descripción/Estructura de la Implementación

```
estudiante@NGEN498:~/Taller_02$ tree
.
├── datos
│   ├── archivo00.txt
│   └── archivo01.txt
├── Implementacion
│   ├── lectura.c
│   └── suma.c
├── Interfacez
│   ├── lectura.h
│   └── suma.h
├── main.c
├── Makefile
└── taller_procesos
```

1. Directorio principal

Contiene los archivos base del sistema y la raíz de compilación.

- **main.c:** Archivo principal del proyecto. Contiene la lógica general del programa, que coordina la lectura de los archivos de entrada, la creación de procesos, el cálculo de sumas parciales y la comunicación de resultados a través de un único pipe. Este archivo centraliza la ejecución del sistema, integra las funciones declaradas en las cabeceras del directorio Interfaces e implementadas en el directorio Implementación.
- **Makefile:** Script de automatización encargado de compilar, enlazar, ejecutar y limpiar el proyecto. Define reglas para construir el ejecutable taller_procesos. Su objetivo es garantizar una compilación estructurada y del sistema.

2. Directorio /Implementación

Contiene las implementaciones funcionales que realizan las operaciones fundamentales del programa.

- **lectura.c:** Implementa la función leer_enteros, la cual abre un archivo de texto lee una cantidad de números enteros. Incluye validaciones de apertura de archivo, reserva de memoria y control de lectura.
- **suma.c:** Contiene las funciones encargadas de realizar las operaciones sobre los arreglos cargados desde los archivos. Implementa suma_array que calcula la suma de los elementos de un arreglo, y suma_doble que combina las sumas de dos arreglos distintos reutilizando la función anterior.

3. Directorio /Interfacez

Incluye los archivos de cabecera (.h) que definen las interfaces de los módulos de implementación. Estos archivos permiten la comunicación entre los distintos componentes del taller.

- **lectura.h:** Declara la función `leer_enteros`, en cualquier otro que requiera operaciones de lectura de datos desde archivos. Su objetivo es establecer una interfaz para la lectura y carga de información en memoria dinámica.
- **suma.h:** Declara las funciones `suma_array` y `suma_doble`, sus parámetros y tipos de retorno. Estas definiciones permiten que las funciones de cálculo sean invocadas desde otros módulos sin necesidad de acceder directamente a su implementación.

4. Directorio /datos

Almacena los archivos de entrada que contienen los arreglos de números enteros utilizados en las pruebas del sistema.

- **archivo00.txt:** Contiene una secuencia de números enteros consecutivos del 1 al 9, usados como datos de entrada para el primer arreglo A.
- **archivo01.txt:** Contiene una serie de valores en múltiplos de diez (10, 20, 30, hasta, 100), que sirven como datos para el segundo arreglo B.

Funciones Clave

- **leer_enteros:** Lee un archivo de texto y carga los valores enteros en un arreglo dinámico. Incluye verificaciones para garantizar la validez de los datos leídos y el correcto uso de la memoria.
- **suma_array:** Calcula la suma de todos los elementos de un arreglo. Esta función es utilizada por otros módulos como componente básico para operaciones de mayor complejidad.
- **suma_doble:** Combina las sumas de dos arreglos diferentes utilizando internamente la función `suma_array` para cada conjunto de datos.

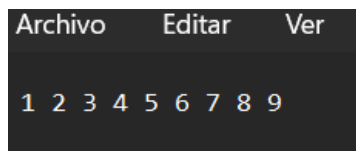
Plan de Pruebas

Propósito

- Verificar que el programa del taller lee correctamente los enteros desde dos archivos.
- Calcula `sumaA`, `sumaB` y `sumaTotal = sumaA + sumaB`
- Comunica resultados por un único pipe desde procesos hijo/nieto al proceso padre

Casos de prueba funcionales

En el archivo00.txt se tienen los siguientes datos:



```
Archivo  Editar  Ver
1 2 3 4 5 6 7 8 9
```

En el archivo01.txt se tienen los siguientes datos:

Archivo Editar Ver

|10 20 30 40 50 60 70 80 90

1. Caso

$$\text{sumaA} = \text{ExpA} (5) = 5*6/2 = 15$$

$$\text{sumaB} = \text{ExpB} (5) = 5*5*6 = 150$$

$$\text{sumaTotal} = 15 + 150 = 165$$

Salida esperada

sumaA=15

sumaB=150

sumaTotal=165

```
estudiante@NGEN498:~/Taller_02$ ./taller_procesos 5 datos/archivo00.txt 5 da
tos/archivo01.txt
sumaA=15
sumaB=150
sumaTotal=165
```

2. Caso

$$\text{sumaA} = \text{ExpA} (5) = 15$$

$$\text{sumaB} = \text{ExpB} (4) = 5*4*5 = 100$$

$$\text{sumaTotal} = 15 + 100 = 115$$

Salida esperada

sumaA=15

sumaB=100

sumaTotal=115

```
estudiante@NGEN498:~/Taller_02$ make run
./taller_procesos 5 datos/archivo00.txt 4 datos/archivo01.txt
umaA=15
umaB=100
umaTotal=115
```

Conclusión plan de pruebas:

Los dos casos validan el comportamiento del sistema frente a distintos tamaños de lectura, la correctitud del cálculo en cada proceso y la integridad de la comunicación a través de un único pipe. La evidencia respalda que el objetivo funcional se cumple:

Lectura segmentada por N, sumas correctas y agregación final consistente.

```
estudiante@NGEN498:~/Taller_02$ make clean
rm -f taller_procesos main.o Implementacion/lectura.o Implementacion/suma.o
estudiante@NGEN498:~/Taller_02$ make taller_procesos
gcc -IInterfacez -c main.c -o main.o
gcc -IInterfacez -c Implementacion/lectura.c -o Implementacion/lectura.o
gcc -IInterfacez -c Implementacion/suma.c -o Implementacion/suma.o
gcc main.o Implementacion/lectura.o Implementacion/suma.o -o taller_procesos
estudiante@NGEN498:~/Taller_02$ ./taller_procesos 5 datos/archivo00.txt 5 da
tos/archivo01.txt
sumaA=15
sumaB=150
sumaTotal=165
estudiante@NGEN498:~/Taller_02$ make run
./taller_procesos 5 datos/archivo00.txt 4 datos/archivo01.txt
sumaA=15
sumaB=100
sumaTotal=115
estudiante@NGEN498:~/Taller_02$ |
```

Conclusiones

En conclusión, el taller fue exitoso en cumplir con los objetivos establecidos. Implementamos con éxito un sistema que:

- Realiza la lectura de archivos de texto y carga los datos en memoria.
- Calcula la suma de los elementos de los arreglos utilizando procesos concurrentes.
- Utiliza comunicación entre procesos mediante pipes para pasar resultados al proceso padre.

El ejercicio demuestra la aplicabilidad de los conceptos de procesos concurrentes, comunicación interprocesos y gestión de memoria dinámica, fundamentos esenciales dentro de la materia de sistemas operativos.

También, el taller evidencia la importancia del manejo cuidadoso de los errores para garantizar la estabilidad del sistema. La implementación refleja una comprensión sólida de las funciones `fork()`, `pipe()` y del uso correcto de la memoria dinámica.